Proceedings of the Second International Conference on Inventive Systems and Control (ICISC 2018)
IEEE Xplore Compliant - Part Number:CFP18J06-ART, ISBN:978-1-5386-0807-4; DVD Part Number:CFP18J06DVD, ISBN:978-1-5386-0806-7

Independent Verification and Validation strategy of Real-Time EXecutive (REX) Software of Onboard Computers of Satellite Launch Vehicle

Shinni Sankar K, Anil Abraham Samuel, Jayalal N, Gopalakrishnan T, B Valsa
ISRO, Vikram Sarabhai Space Center,
Thiruvananthapuram, India
shinni_sankar@vssc.gov.in, anil_abraham@vssc.gov.in, t_gopalakrishnan@vssc.gov.in, b_valsa@vssc.gov.in

***Abstract* — The most important quality assurance activity in flight software development is techniques used for software verification and validation. When compared with conventional software, testing of real-time systems presents unique and challenging problems since various real-time constraints of the software also need to be verified. The real-time executive software component (REX) of onboard flight software should be tested for the real-time aspects to accurately evaluate the system performance in different implementation scenarios. This paper summarizes the Independent Verification and Validation strategy followed for REX Software.**

Keywords—Real-Time software; Verification and validation; Software testing; Fault Injection tests.

I. INTRODUCTION

The Onboard Computing (OBC) system of a satellite launch vehicle executes mission critical tasks such as Sensor data acquisition, Navigation, Guidance, Control and Mission Sequencing (NGC functions) of the vehicle in flight. The OBC system use fault tolerant tightly synchronized and distributed computers with tasks scheduled in a fixed priority preemptive manner in different computers. The Real-time Executive software (REX) which is the system software of the master computer of the OBC system coordinates the different processors, tasks and resources in real-time. REX schedules execution of different tasks in two periodicities viz. minor and major cycles, establishes communication between tasks/input and output processors, detects anomalies in system operations, initiates recovery procedures and keeps watch on system health.

Verification & Validation techniques have key role in ensuring that the onboard software is correct and meet the requirements. A software error can cause the loss of an expensive mission, or lead to budget overruns and affect the schedules of commercially committed launches. This signifies the importance of having an efficient verification and validation process for critical software systems. To assure that the mission critical software performs its intended functionality even in case of unpredictedevents, it should be established as part of V&V that the software is deterministic and predictable in such scenarios.

In this paper we are describing the verification and validation practices, especially the testing techniques, followed for the evaluation and qualification of REX software.

II. REAL-TIME EXECUTIVE SOFTWARE (REX) –AN OVERVIEW

The REX software of the master computer of the OBC system has two working modes namely monitor mode and flight mode. The REX software has a boot program that takes the computer to a non real-time interfacing mode called monitor mode, when it is powered on. There are interfaces for a user to check the functionalities of the hardware elements of the computer in this mode. Custom defined interface rules based on Avionics Bus communication protocol is used for user interfacing.

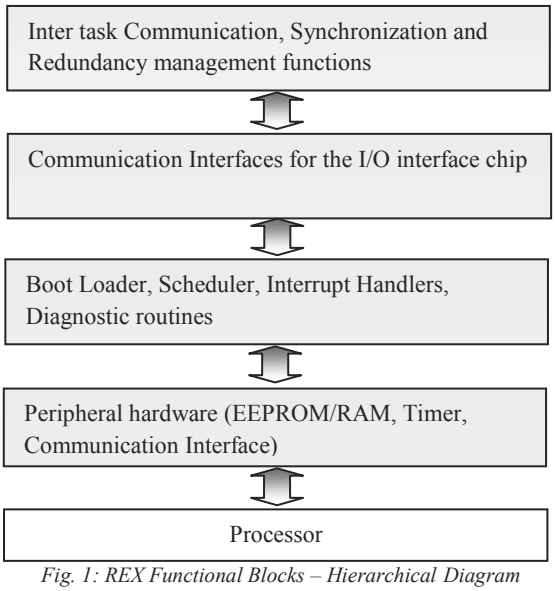

Fig. 1: REX Functional Blocks – Hierarchical Diagram

In flight mode, which is the real-time mode of operation, REX software performs the scheduling of required application tasks in required periodicities. The priorities of these tasks are assigned using a rate monotonic scheduling

policy [5]. In addition REX performs input/output management with other computers in the system, synchronization of all the computers, interrupt handling, redundancy management, initiates system error handling actionsand reconfiguration when encounters various system errors.

III. INDEPENDENT VERIFICATION AND VALIDATION (IV&V) FOLLOWED FOR REX SOFTWARE

Testing and debugging of real-time REX software is one of the most challenging task in the quality assurance of flight software. As the onboard flight software is mission critical, independent verification and validation (*IV&V*) is carried out, i.e. verification and validation of the software is performed by an independent agency other than the design team. When performed by an independent agency tests are designed with a 'test to break' philosophy rather than following a 'test to work' philosophy and there by detect and eliminate defects earlier and eliminate any latent defect that can be a potential candidate for losing a mission.

Hence Independent V&V is meticulously done for all the mission critical software elements used in launch vehicle. The major *IV&V* tasksperformed for REX software are:

- Static Analysis and Code Inspection
- Module Level Tests
- Software Fault Injection Tests
- Integrated Processor Tests
- Closed Loop Simulations Tests

IV. STATIC CODE VERIFICATION

The goal of static code verification is to assess a program without executing it. The major activities carried out are – Static analysis and Code Inspection. In static analysis syntactic, structural and semantic analysis of the software is carried out using automated software tools. The analysis gives details on data declaration, initialization and usage, function calls, structure charts, control flow graphs and software metrics such as complexity measures and size information. The static program analysis includes complexity analysis, structural analysis and data-flow analysis. And the analysis outputs are used to assess whether the software comply with the stipulated design and coding guidelines and helps in detecting and removing many defects earlier. It gives insight about the quality of a software product and localizes error-prone positions in the software.

Code inspection helps to find out design and implementation errors. During Code Inspection, correctness of the implementation with respect to the software design is established and uncovers missing/wrong/extra code present in the software.As the errors are found out early in the life cycle, the corrections can be done before the testing and the testing effort is considerably reduced. The inspection team consists of two or three members. An experienced software engineer with domain knowledge as team leader and one or two software testing person are involved in the inspection team. The team members will be part of the software verification activity through their participation in the software peer reviews of software specification onwards.

Apart from verifying the code w.r.t the software design, various real-time aspects of the REX software are specifically considered during inspection, which include:

- Analyzing the scheduling precedence, and data dependencies across tasks
- End-to End Verification, from Input Sensors to onboard computer and to output Actuators.
- Timing constraints specified for various functions
- Adequacy of time outs provided while retrieving data from other onboard processors
- Occurrence of events that occur in various onboard processors and their effect on the system
- System synchronization and error handing logics

V. REX SOFTWARE TESTING

Dynamic testing of the software is an imperative process in the software life cycle. The combinations of possible input conditions are prohibitively large in most software programs. Hence exhaustive testing of software, covering all possible input combinations, is effectively impossible. So the test cases are judiciously selected to ensure test coverage for conditions, decisions andall statementsin the code as well as all functional requirements of the software, exercising all possible valid input and output domains.

Unlike non real-time software where the focus is functional testing, testing of the real-timesoftware demands some additional dimensions to be covered. A single testing methodology alone could not cover all the real-time requirement aspects of software. There is a need to do robustness testing and performance testing, in addition to the functional tests. Also the behavior of the software under various asynchronous events such as interrupts and exceptions shall be carefully evaluated. This multi dimensional functional-robustness-performance-behaviortesting strategy is essential for achieving the test coverage of the REX software.

A. Functional Tests

The functional tests are carried out at module level [6] and at integrated level through closed loop simulations.

1) Module level testing

In module level, the traditional black box (requirement based) test cases and white box (structural coverage based) test cases forms the basis for test case design. The test cases which exercise functional requirements of the program uncover deviations of the software from its specification; typical errors found out includes incorrect or missing functions, interface errors, initialization or termination

errors. Structural test cases which are designed based on the control flow structure of the software ensures test coverage forprogram statements, basis paths, loops, conditions and logical decisions in the software and uncovers structural anomalies.

The module level test setup consists of a host computer having a cross compiler, cross assembler, a linker for the target and the processor simulator. The test cases designed for REX software are run onthe simulator, where input acquisitions from other hardware modules are statically simulated. When the software modules are tested individuallyin module level, interfaces and many other factors that will interact or affect the real-time behavior of the software do not come into picture. Hence, only logical and *non*-timing errors are uncoveredin module level tests and the end-to-end system do not get tested here which is a lacuna in this testing level.

2) Integrated Simulation Tests

Since flight software performancedepends onphysical conditions experienced by the launch vehicle and behavior of many subsystems, validation of the onboard software in simulated flight conditions is essential.

Validation of the OBC is carried out in simulated flight runs in open loop and closed loop modes with sensors and actuators are included in the loop. The flight software is interfaced with 6D-trajectory simulation software to simulate flight environments, vehicle propulsion systems, aerodynamics, vehicle characteristics, vehicle dynamics and vehicle subsystem dynamics such as inertial sensor and Control Power Plant (CPP) dynamics. Appropriate mathematical models are used for simulating the characteristics of these subsystems.

In closed loop tests, the outputs from the package/system under test are fed back to the 6D trajectory simulation system for computing the next cycle inputs. The system level tests are carried out in different simulation test beds with different levels of nearness to flight conditions and the main closed loop simulations are Onboard processors in loop Simulations (OILS), Hardware-in-Loop Simulations (HLS) and Actuator -in-Loop Simulations (ALS). In OILS, the software is stressed for a wide range of input and output conditions, exercising very benign to high stress environments. Actual flight sensor hardware and control power plant elements are tested in HLS and ALS test runs. These simulation results give the flight usage clearance for software and various hardware elements.

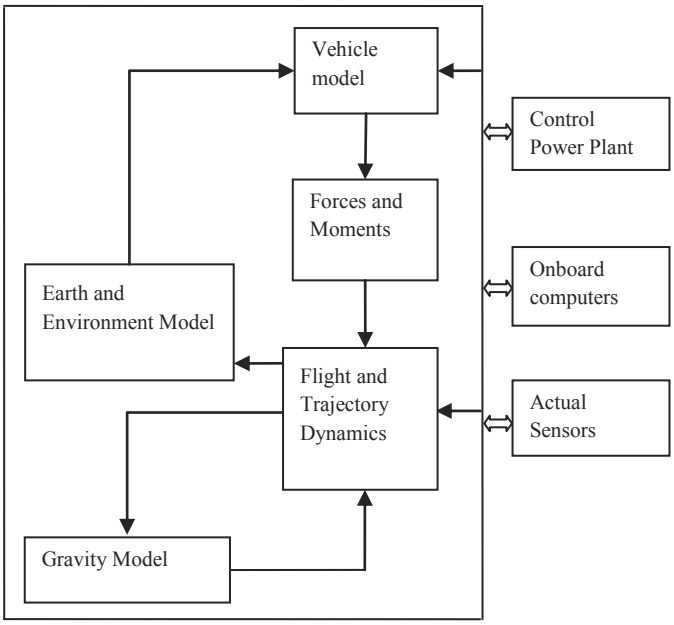

Fig. 2: Elements in Simulation Tests

B. Robustness Testing

While testing the robustness aspects, the goal is not only to test the completeness of the software requirements and the correctness of the results, but also to test whether the software system is robust enough to handle invalid inputs and events. System errors may be time dependent, only arising when the system and controlled environment are in a particular state that may be impossible to reproduce. Two different validation techniques are followed for the robustness evaluation-Integrated Processor Testing (IPT) and Software Fault Injection Testing (SFIT).

1) Integrated Processor Testing (IPT)

Objective of Integrated processor tests (IPT) is to exercise flight software residing in various onboard computers in open loop mode which validates hardware-software interface, link error handling, redundancy management and system synchronization aspects. This is the test bed where the integrated system of various onboard processors is subjected to verify the timing and fault tolerance aspects. IPT test cases are worked out based on the failure detection and isolation logic implemented in the onboard system. The test cases cover OBC system input interface failures, communication link failures between various subsystems, package power failure, Avionics Communication bus failure, clock drift scenarios and processor failures. The software robustness under these failures is evaluated. For integrated processor testing for the flight software for a mission, around 500 test cases were run to validate the subsystems.

System synchronization Tests- A typical robustness evaluation test in IPT

The onboard processing elements in the launch vehicle are in dual redundant hot standby configuration, where each prime and redundant module is connected to both prime and redundant avionics communication buses. The input acquisition units in the NGC system perform sensor data acquisition and then send it to the main onboard computer (OBC-Master) via messages through thebus. Similarly the output units in the NGC system receive the commands from OBC-Masterthrough bus and distributes to the control actuators and drivers. In this distributed architecture, input acquisition, data processing, output posting and failure detection & reconfiguration logic are shared among different processing elements and each processor maintains its own clock.

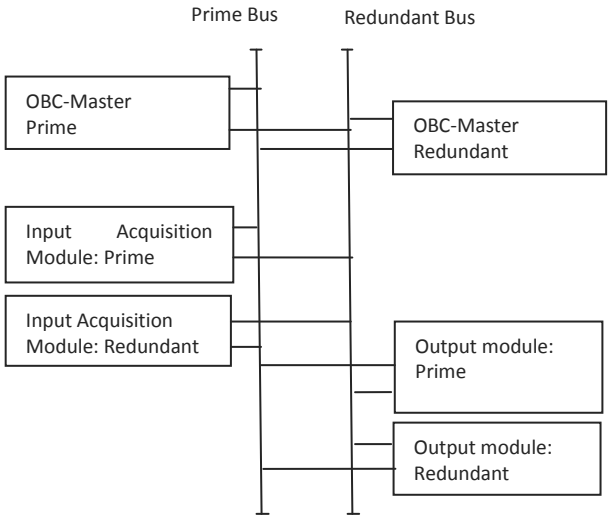

Fig. 3: System Configuration – Onboard processing Elements

To preserve the logical and temporal ordering of the tasks in this system, clocks and message passing between various processors should be in tight synchronization. The OBC master is the communication bus controller as well as synchronization master which with other processing elements align their minor cycle boundary at flight reset. Subsequently, clocks of input/ output processing elements in onboard systems should be in tight time synchronization with OBC master clock during the entire flight duration to satisfy the controllability requirements and to facilitate smooth reconfiguration to redundant modules in case of any failures.

In each subsystem, the skew w.r.t sync message reception from main on-board computer is measured and if the clock drift is greater than predefined lower limit, timer correction is done to sync with closer OBC. If the skew measured is above an upper limit, it indicates that clocks have drifted beyond correctable limits and hence no correction is applied. Since time stamp of sync message reception is the input for skew calculation, sync message reception within an expected window in every minor cycle is essential. Hence, to test the logic, link failure simulation cases between OBCs are carried out.

Skew window limits are calculated considering worst case clock accuracy specification of onboard computer. If the timer drifts beyond a range, beyond possible worst case clock drift, then synchronization of redundant OBC with prime will be disabled.To test this, positive and negative clock drifts are given upto possible and beyond upper limit values.

2) Software Fault Injection Testing (SFIT)

SFIT is a dynamic testing technique to evaluate the robustness of the software to handle internal or hardware related errors, and is performed by intrusive methods.When the flight software in one OBC is tested,other computer packages in the system are simulated in terms of all activities as in flight. Test cases are designed to test the built-in fault handling features such as timer failure, task incompletion,memory corruptions, communication failure, arithmetic errors, computational overflows and processor failure. Each test constructed creates a realistic fault in the system that when manifested as an error, exercises a corresponding fault tolerant feature in the REX software [2].

With fault injection tests, defects in the error handling implementation within flight software were pointed out and corrected. These errors found out were residing in the fault handling paths that were not triggered in normal integrated runs including closed loop simulations. Fault injection is a practical approach to perform stress testing, i.e., raising conditions to trigger rarely executed software operations such as error handling and recovery features that otherwise will not be tested through functional tests.

C. PERFORMANCE TESTS

Since the correctness of real-time software depends on the time at which inputs are present to the system and outputs are produced. For the predictable system performance ofreal-time software, systematic strategies are adopted to verify:

- Response time requirements
- Latencies
- Execution Time of Individual Tasks
- Timing Margin
- Memory Usage assessment
- Communication Bus Load

1) Response Time Tests

The main features seen when analyzing the response time requirements are:

- Cycle time and Task execution time
- Interrupt Latency
- Context Switching Time

2) Cycle Time and Execution Time Evaluation

Timing evaluation is a difficult element in real-time testing. When a real-time system is considered, a correct output that is *not produced within a specified time interval* potentially constitutes a failure.The onboard computer has a well definedcycle time requirement.Sensing a set of inputs describing launch vehicle transitional and rotational dynamics, processing it and generating the commands for the actuation elements should happen within this cycle time. This is the most important performance requirement in NGC system which cannot be violated. If it is violated then proper fault handling procedures have to be initiated in the onboard software and indicated to all the participating computers in the system, so that the violated computer can be isolated.

The onboard software is inbuilt with monitors that monitor the completion of tasks in time. If any violation happens then in the next timer interrupt this is detected and fault handling action is evoked.

It should be ensured that the software handles any timing deviationin the system, that is, outcome that do not occur according to the timing specifications due to planned or unplanned events in the system. The timing behavior of the system is measuredin terms of the execution time of the tasks; the delays encountered due to various events in the system and scheduling delays such as context switchingand interrupt latency.

For execution time evaluation:

a. The task entry and exit times are noted and the execution time of a task is calculated from these figures. Here, the hardware timer generating interrupt every 5 ms is initially loaded with corresponding value and is reloaded whenever the counter decrements to 0. Value of this timer is monitored at a task's scheduling point and exit point. Since a task can be in execution state over multiple interrupt cycles, different execution time slices for that task at each ISR entry is accounted in execution time calculation.
b. Instruction counting in target simulator: With the execution trace and the number of machine cycles taken for each op code, the execution time taken can be calculated, even for a standalone module. This has the drawback that it cannot really give the delays caused by hardware such as memory read/write wait-states inserted. Hence, for assessing real-time performance, this method cannot be used.
c. In initial development phase, to get an assessment of instruction timing, digital scopes which monitor particular signatures in processor's address and data bus is used to probe the execution time.

The execution time taken for each task is measured by the scheduler, which along with the other timing factors give the execution time margin in the system. During initial development phase, these measures provide potential information to the designer about optimizing the software to improve the program efficiency.

3) Memory Usage Assessment

Additionally, since the onboard processor memory is bounded, it is a major input constraint as far as the software implementation is considered; sizing parameters of the onboard software should be estimated. There exists stringent design and coding guidelines which enforces safe-subset rules to the software to ensure that it will be optimized in terms of size and execution time and it will not exhibit any dynamic execution behavior. As soon as the code is stable and ready for integration, the size of each executable component, programand data memory used andstack usageare measured by running the software, as part of software qualification. Special load/stress tests are run as part of simulation stress cases to determine the maximum memory used, as well as to obtain timing and throughput parameters.

VI. BEHAVIOURAL TESTS

A. Interrupt Tests

To ensurethe deterministic performance, it is important to make sure that all interrupt priorities and responses are well defined and handled by REX software. It has to be ensured that usage of interrupts will not lead to priority inversion or prevent a high priority or safety critical task from completing.The system should be ableto stack incoming interrupts to prevent their loss; they should be serviced as per the priorities defined. A low-priority process shall not interrupt a high-priority process and change critical data. While performing interrupt analysis, these aspects are tested.

As far as REX software is considered, there are three defined interrupts: Double bit memory error interrupt having highest priority, Timer Interrupt and Single bit memory error interrupt; where the timer interrupt manages the real-time scheduling and the other two are generated by an external error-detection-and-correction circuitry for a processor RAM corruption (SECDED Errors). Nesting of interrupts is not currently supported by the software.

As part of interrupt testing, offline routines are used to measure interrupt latency and to exercise error-detection-and-correction on all memory cells in processor RAM. In the timer test routine, the ISR will be entered for a large predefined count and the actual number of ISR entry is measuredto verify the interrupt servicing. In flight software, occurrence of task completion and execution frame boundaries are identified within timer interrupt ISR, hence it contains error handling paths for identification of deadline misses and computational errors for each process. Specific test cases are designed and executed as part of SFIT to inject such errors in scheduling points, namely deadline misses, computational overflow, processor clock errorsand it is verified that ISR executes the appropriate error handling action towards these errors.

Critical sections in the core modules are protected from race conditions by disabling the interrupts for such segments. When performing interrupt analysis, program segments where interrupts are disabled such as critical sections or timing-critical areas arethoroughly inspected. Such segments should be kept as short as possible and should be entered as infrequent as possible; there should not be any possibility of indefinite loops and servicing of any buffered interrupts should be as per the priority.

B. Task scheduling

REX executes a preemptive, static priority based rate-monotonic scheduler during flight[5]. Under rate monotonic prioritization, tasks are assigned priority monotonically as an increasing function according to their rate. The mission critical tasks performed in different periodicities have hard real-time deadlines. These tasks should handle asynchronous inputs from other processors. Exact execution points at which the inputs will be presented to the tasks is highly unpredictable.

Primary aim of schedulability analysis is to ensure predictability in task scheduling i.e. whether all deadline requirements are met for each of the constituent tasks in existing scheduling policy. Individual testing of each task uncovers errors in logic and function but not timing or behavior. Verifying whether the software satisfies the temporal requirements and precedence constraints, with the finite set of processing resources is the main criteria when testing a scheduler.

Handling deadline misses is done in the flight software by using the run time information set by each task. A deadline is violated when a task is not completed within the stipulated time frame, which is termed as *task incompletion* and can arise due to undetected software defect or by data corruption or due to any particular input data combination, causing infinite loops or algorithms failing to converge or a change in the program flow. To test the software for deadline misses, test scenarios that maximize the chances of deadline misses within the system are included as stress cases in validation tests. Here the input conditions are given such that the software will take the worst case execution path, and task completion is pushed as close as possible to the deadline. While testing task priorities of the real-time tasks, it should be verified that all time-critical and precedence oriented events will be assured of execution and that each task complete within its guaranteed response time. In a nominal run, the normal executions sequences are always getting ensured. As part of software fault injection tests, it is ensured that *all paths* within scheduler code are exercised. Specifically, in SFIT[2], test cases were designed to cover the error handling paths dealing with many unanticipated scenarios and ensures that deadlines are met even under additional error handling paths. Also, specific cases are used to trigger task incompletion and validate the fault handling features.

VII. CONCLUSION

This paper summarizes various verification and validationschemes / testing approaches adopted for real-time software used in Satellite launch vehicles of Indian Space Research Organization(ISRO). For software validation, starting from unit tests which ensure requirement and path coverage tests at module level, testing is progressed to integrated level system tests, ensuring behavioral and performance assessment of the software. For each mission, various system levelsimulation tests with mission specific profile and onboard elements/hardware in loop are performed to validate software or data changes, giving confidence for final software clearance for flight.

REFERENCES

[1] Laplante and Philip AS. ‘Real-Time Systems Design and Analysis, Publisher, John Wiley & Sons Edition 3(2005).’

[2] Anil Abraham Samuel, Jayalal N., Valsa B., Ignatious C.A., and John P. Zachariah. ‘Software fault injection testing of the embedded software of a satellite launch vehicle.’ - IEEE Potentials, Sept 2013

[3] Manju C.R., Josna Susan Joy, Vidya L., Sheenarani I., Jayalekshmy L., KesavabrahmajiKaruturi, Sheema E, Syamala S, R. Paramasivam, Abdul Shukkoor A, Mookiah T.

[4] ‘Mission Management Computer Software for RLV-TD’ - Institute of Engineers Journal.

[5] Anil Abraham Samuel, Jayalal N, Gopalakrishnan T, Valsa B. ‘Best Practices in ISRO for the Satellite Launch Vehicle Flight Software Qualification.’ Presented in Embedded Safety and Security Summit (ESSS June 2016). April, 2016.

[6] Lui, C.L. and J.W. Layland. 'Scheduling Algorithms for Multiprogramming in a Hard Real-TimeEnvironment.’ Journal of ACM Vol 20(1), 1973: 46-61.

[7] Beizer B. Software testing techniques. Van Nostrand Reinhold, 1990